USIGS Common Object Specification

National Imagery and Mapping Agency
United States Imagery and Geospatial System

Release Date: 22 July, 1997
Version 1.0

Acknowledgments

Many individuals and organizations provided support and technical contributions to this work, Individuals from numerous government agencies, contractor organizations and vendors contributed significantly to the development of this specification. We acknowledge these contributions and hope that these individuals and organizations will continue to actively support future updates and extensions. Thanks in advance.

Revision History

- Initial release of UCO Specification. Derived from earlier work on Imagery Access Services and Geospatial and Imagery Access Services
  Version 1.0 - Released for NCCB submittal 22 July 1997

- USIGS Common Object Specification Version 1.1 - Update to incorporate responses and comments from additional interface prototyping tests - Nov. 1997 (TBR)

- Regular updates at approximately six month intervals or as required to support additional interface definition efforts.

Preface

This document defines common interfaces and datatypes that are expected to used by many other United States Imagery and Geospatial System (USIGS) interface specifications. The intent of this specification is to document the interfaces, datatypes and error conditions that are expected to most commonly occur or be most broadly applicable across the USIGS architecture. The use of these common definitions will support interoperability among the various interface specification in the USIGS architecture.

This specification was prepared consistent with industry practices and is modeled after those being prepared by the Object Management Group (OMG) industry consortium. This approach is consistent with guidelines and direction established by the NIMA Common Imagery Interoperability Working Group (CIIWG).

Table Of Contents

1. Overview

The USIGS Common Object Specification (UCOS) is a critical element to support interoperability in the USIGS architecture. The purpose of the UCOS is to define in a single place the interfaces, data types and error conditions that must be shared by multiple specifications in USIGS. By defining these shared elements in a single place, redundant (possibly non-interoperable) re-definitions of the same concept in multiple specifications are prevented. It is necessary for the effective use of this specification, that the elements defined here be used whenever and where ever appropriate and that all new shared elements identified by specifications be incorporated in later versions of this specification.

2. UCO Elements

2.1 Overview

All UCOS element definitions are enclosed within the UCO module:

```
module UCO
{

... all UCO elements...

};
```

2.2 UCO data types

2.2.1 General data types

2.2.1.1 NameList, NameValue & NameValueList

```
struct NameValue
  {
    string name;
    any value;
  };

typedef sequence < NameValue > NameValueList;

typedef sequence < string > NameList;
```

These three data types are used as containers for generic identifiers (“Names”) as well as the association of identifiers with a value.

The NameValue structure is used to associate an identifier defined as a type string with a value defined as a type any. The type any is used as a container to hold a value of any system or user defined type.

The NameValueList is used as an ordered list of NameValue pairs. This structure does not define any explicit relationship amongst the elements of the list except their order in the sequence.

The NameList is used as an ordered list of identifiers. This structure does not define any explicit relationship amongst the elements of the list except their order in the sequence.

2.2.1.2 FileLocation & FileSet

```
struct FileLocation
  {
    string user_name;
    string password;
    string host_name;
    string path_name;
    string file_name;
  };

typedef sequence <FileLocation> FileSet;
```

These two structures are used to identify individual and collections of files and file locations in a file system.

The FileLocation structure is used to define an individual file or file location as well as provide access control information needed to access that file or location. The system on which the file or location resides is defined by supplying a host name or IP number, as a string, in *host_name* . The location of that file on the specified system is defined by supplying, as a string, an absolute path to the directory in which the file resides. The syntax for this string uses the UNIX path specification i.e. "\" is used as a delimiter and all paths begin with a delimiter indicating the root directory. The identifier for the specific file is identify by supplying, as a string, the file name in *file_name* . If the FileLocation structure is being used to specify a directory, the full path including the desired directory will be included in *path_name* and *file_name* will be NULL. The information needed to perform access control for the specified file is in the elements *user_name* and *password* .
An identifier for a user authorized to access the requested file or location is supplied as a string in *user_name* . The password that corresponds with that user identifier is supplied as a string in *password* . The values for these access control elements for public ("anonymous") access are implementation dependent.

The structure FileSet is used to represent a collection of files or file locations. Each file or location in the collection is a complete FileLocation structure, (i.e. there are no assumptions about the files or locations or their access control information with respect to each other). This structure does not define any explicit relationship amongst the elements of the collection except their order in the sequence.

2.2.1.3 Date, Time & AbsTime

```
struct Date
  {
    unsigned short year;
```

```
    unsigned short month;
    unsigned short day;
  };

struct Time
  {
    unsigned short hour;
    unsigned short minute;
    float second;
  };


struct AbsTime
  {
    Date date;
    Time time;
  };
```

These structures are used to represent points in and periods of time. The Date structure is used to represent a specific day. The year of that day is specified as an unsigned short in *year* . This must be the full specification of the year, using the last two digits (i.e. “97” for “1997”) is considered non-compliant. The month of this day is specified as an unsigned short in *month* with a value between 1 and 12, where the value 1 represents January and the value 12 represents December. The specific day is identified as an unsigned short in *day* . To represent a general calendar day of the year, (i.e. Independence Day is 4 July) the year value shall be 0 (zero).

The structure Time can be used to represent both absolute and relative time. In order to represent absolute time, this structure is used with the Date structure to establish a point of reference (See the AbsTime structure defined below). To represent relative time or a length of time, the number of hours in the period is supplied as an unsigned short in *hour*, the number of minutes supplied as an unsigned short in *minute* and the number of seconds supplied as a float in *second*.

The structure AbsTime is used to represent an absolute point in time. It is composed of a Date structure (see above) and a Time structure. The Time structure is considered to represent the time in Military time (i.e. 24 hour clock).

2.2.1.4 EmailAddress

```
typedef string EmailAddress;
```

This structure serves to hold a definition of a complete email address. The syntax for this string is of the form user_name “@” host_name, where *user_name* is an identifier for a user or account on system *host_name* .

host_name is an identifier for the target system. It can be in the form of a name or IP address.

2.2.2 Vector-like data types

The following data types are used to represent geospatial and geometric elements.

Note: These data types may be replaced in the future with definitions from the Open GIS Consortium (OGC), a standards forum in the GIS community. The OGC data types were not available at the time of development of this specification. When those data types definitions do become available, they will be reviewed and considered for inclusion in this document to replace the data types defined in this section.

2.2.2.1 Coordinate

```
struct LongCoord {
    long x;
    long y;
    long z;
  };

struct DoubleCoord {
    double x;
    double y;
    double z;
  };

enum CoordinateType
  {
    TypeDouble, TypeLong
  };

union Coordinate switch (CoordinateType)
  {
    case TypeDouble:
      DoubleCoord d_coord;
    case TypeLong:
      LongCoord l_coord;
  };
```

The Coordinate data type serves as a basis for all composite geospatial data types in UCOS. The intent of Coordinate is to represent a single three dimensional point in a coordinate system. The definition of the coordinate

system being used for a specific Coordinate is defined by its usage or other structures containing the Coordinate, not the Coordinate structure itself.

A Coordinate can be of one of two forms: TypeLong or TypeDouble. TypeLong Coordinates are used for coordinate systems which have dimensions requiring integer values. TypeDouble Coordinates are used for coordinate systems which have dimensions requiring floating point values. The values of the three dimensions of the point to be represented are placed in the *x* ,*y* and *z* elements of the LongCoord or DoubleCoord type as appropriate. The mapping of coordinate system dimension to *x* ,*y* and *z* elements is coordinate system dependent.

2.2.2.2 LineString

```
typedef sequence < Coordinate > LineString;
```

The LineString data type is used to represent a piece-wise linear path through three dimensional space. The path is constructed by connecting the individual Coordinates with line segments in the order they appear in the sequence. The order of the Coordinates in the sequence is used only to define the connections of the Coordinates, it does not necessarily imply a direction for the line string nor a temporal sequence (trajectory).

2.2.2.3 Rectangle

```
struct Rectangle
  {
    Coordinate ul;
    Coordinate lr;
  };
```

The Rectangle structure is intended to provide a simple definition of an area on a surface. It is defined by supplying two Coordinates indicating the opposing corners of the rectangle of interest. By convention, for coordinate systems describing the surface of the Earth, the *ul* element will indicate the Northwest corner and the *lr* element will indicate the Southeast corner.

2.2.3 Raster-like data types

2.2.3.1 Buffer

```
enum BufferType
  {
   OCTET_DATA, CHAR_DATA, SHORT_DATA, USHORT_DATA, LONG_DATA,
   ULONG_DATA, FLOAT_DATA, DOUBLE_DATA
  };
```

```
union Buffer
 switch (BufferType)
   {
    case OCTET_DATA:   sequence < octet >          octet_data;
    case CHAR_DATA:    sequence < char >           char_data;
    case USHORT_DATA:  sequence < unsigned short > ushort_data;
    case SHORT_DATA:   sequence < short >          short_data;
    case ULONG_DATA:   sequence < unsigned long >  ulong_data;
    case LONG_DATA:    sequence < long >           long_data;
    case FLOAT_DATA:   sequence < float >          float_data;
    case DOUBLE_DATA:  sequence < double >         double_data;
    default:           sequence < octet >          other_data;
   };
```

2.2.3.2 SimpleGSImage

```
struct SimpleGSImage
  {
    long width;
    long height;
    Buffer pixels;
  };
```

The SimpleGSImage is intended to provide a basic definition of a simple grayscale image. This simple image has two attributes *width* and *height* which define the layout of the pixel data contained in Buffer *pixels.* The Buffer which hold the pixel data is of length width * height. Each value in the Buffer indicates the value of an individual pixel, beginning at the upper left corner of the image and continuing across the top of the image for *width* pixels.

2.2.3.3 SimpleCImage

```
struct SimpleCImage
  {
    long width;
    long height;
    Buffer redPixels
    Buffer  greenPixels;
    Buffer bluePixels;
};
```

The SimpleCImage is intended to provide a basic definition of a simple color image with three bands. This simple image has two attributes *width* and *height* which define the layout of the pixel data contained in the three Buffers. The nth color pixel of this image is defined by the triplet composed of the nth element of each of the three Buffers. Each Buffer is of length *width* * *height* . Each element of the Buffer indicates the value of a color component (red,

green or blue) of an individual pixel, beginning at the upper left corner of the image and continuing across the top of the image for *width* pixels.

2.2.4 State & Status

```
enum State
  {
    COMPLETED, IN_PROGRESS, ABORTED, CANCELED, PENDING, OTHER
  };

struct Status
  {
  State completion_state;
  string status_message;
  NameValueList details;
  };
```

The Status and State structure are intended to represent the current condition of a process, request or other system element.

The enumeration State defines 6 (six) identifiers (“states”) for the condition of a process, request or system element. Each state is also defined as a TERMINAL or NON-TERMINAL state. A process in a TERMINAL state will remain in that state until action is taken that causes it to change state. A NON-TERMINAL state will eventually change to a TERMINAL state without any further action being taken. The identifiers and the corresponding condition are defined as follows:

State	Description	Terminal ?
COMPLETED	All requested processing has completed successfully	Yes
IN_PROGRESS	Still processing, no error or abnormal conditions yet encountered	No
ABORTED	Processing has stop due to an error or abnormal condition	Yes
CANCELED	Processing has been stopped by request	Yes
PENDING	Processing has not yet begun or has temporarily been halted	No
OTHER	A condition other then those described above	Yes

The structure Status is used to describe the details of the current condition of a process, request or system element. The Status structure is composed of three elements: *completion_state* a State (see above) indicating the current condition of the process, request or system element, *status_message* , a string containing a human readable message that further amplifies or clarifies the State and *details* a NameValueList that contains any

attributes or other information that describe or refine the current condition of the process, request or system element being described. The names and values contained in the NameValueList are defined by the context of use and the type of process, request or system element being described. They will be defined by the appropriate profile of the specification using the Status structure.

2.3 UCO interfaces

There are currently no interfaces defined in the UCO specification. In the future, interfaces that define capabilities common to all or required by many of the USIGS interface specifications will be standardized and defined in the UCO specification.

2.4 UCO Exceptions

There are currently no exceptions defined in the UCO specification. In the future, exceptions common to more then one USIGS interface specification will be standardized and defined in the UCO specification.

3. Appendix A: UCO IDL

```
//************************************************************
**
//*
//*                     The USIGS Common Object Specification
//*
//*
//*     Description: Defines fundamental data types and
//*     interfaces to be used by other specifications to
support
//*     interoperation across independently designed
interfaces.
//*
//*
//*
//*     History:
//*     Date        Author      Comment
//*     -----       --------        ------------
//*     15 May 97   D. Lutz     Initial release for review
//*     2 July 97       D. Lutz     Released for TEM review
//*     11 July 97  D. Lutz Changes based on 2 July TEM Comments
//*
//*     Notes
//*     -------
//*     NONE
//*
//*
//************************************************************
**********
// The USIGS Common Objects

module UCO
{
// Generic data types

  struct NameValue
    {
      string name;
      any value;
    };

  typedef sequence < NameValue > NameValueList;
  typedef sequence < string > NameList;

  struct FileLocation
    {
      string user_name;
      string password;
      string host_name;
```

```
      string path_name;
      string file_name;
    };

typedef sequence <FileLocation> FileSet;

  struct Date
    {
      unsigned short year;
      unsigned short month;
      unsigned short day;
    };

  struct Time
    {
      unsigned short hour;
      unsigned short minute;
      float second;
    };


  struct AbsTime
    {
      Date date;
      Time time;
    };


  typedef string EmailAddress;

// Basic Geospatial  data types

//3D integer coordinate
struct LongCoord {
long x;
long y;
long z;
};

// 3D floating point coordinate
struct DoubleCoord {
double x;
double y;
double z;
};


  enum CoordinateType
    {
      TypeDouble, TypeLong
```

```
    };

  union Coordinate switch (CoordinateType)
    {
    case TypeDouble:
      DoubleCoord d_coord;
    case TypeLong:
      LongCoord l_coord;
    };


  typedef sequence < Coordinate > LineString;

  struct Rectangle
    {
      Coordinate ul;
      Coordinate lr;
    };

// Simple composite geospatial datatypes

enum BufferType
   {
      OCTET_DATA, CHAR_DATA, SHORT_DATA, USHORT_DATA,
LONG_DATA, ULONG_DATA,
      FLOAT_DATA, DOUBLE_DATA
   };

   union Buffer
      switch (BufferType)
      {
         case OCTET_DATA:   sequence < octet >
octet_data;
         case CHAR_DATA:    sequence < char >
char_data;
         case USHORT_DATA:  sequence < unsigned short >
ushort_data;
         case SHORT_DATA:   sequence < short >
short_data;
         case ULONG_DATA:   sequence < unsigned long >
ulong_data;
         case LONG_DATA:    sequence < long >
long_data;
         case FLOAT_DATA:   sequence < float >
float_data;
         case DOUBLE_DATA:  sequence < double >
double_data;
         default:           sequence < octet >
other_data;
      };
```

```
  struct SimpleGSImage
    {
      long width;
      long height;
      Buffer pixels;

};


  struct SimpleCImage
    {
      long width;
      long height;
      Buffer redPixels;
      Buffer greenPixels;
      Buffer bluePixels;
    };


  enum State
    {
      COMPLETED, IN_PROGRESS, ABORTED, CANCELED, PENDING, OTHER
    };

  struct Status
    {
    State completion_state;
    string status_message;
    NameValueList details;
    };



};              // End of module UCO
```

4. Appendix B: Reference OMG Standard IDL

CORBA Standard Exceptions

```
#define ex_body {unsigned long minor; completion_status
completed;}

enum completion_status {COMPLETED_YES, COMPLETED_NO,
COMPLETED_MAYBE};
enum exception_type {NO_EXCEPTION, USER_EXCEPTION,
SYSTEM_EXCEPTION};

exception UNKNOWN ex_body;
exception BAD_PARAM ex_body;
exception NO_MEMORY ex_body;
exception IMP_LIMIT ex_body;
exception COMM_FAILURE ex_body;
exception INV_OBJREF ex_body;
exception NO_PERMISSION ex_body;
exception INTERNAL ex_body;
exception MARSHAL ex_body;
exception INITIALIZE ex_body;
exception NO_IMPLEMENT ex_body;
exception BAD_TYPECODE ex_body;
exception BAD_OPERATION ex_body;
exception NO_RESOURCES ex_body;
exception NO_RESPONSE ex_body;
exception PERSIST_STORE ex_body;
exception BAD_INV_ORDER ex_body;
exception TRANSIENT ex_body;
exception FREE_MEM ex_body;
exception INV_IDENT ex_body;
exception INV_FLAG ex_body;
exception INTF_REPOS ex_body;
exception BAD_CONTEXT ex_body;
exception OBJ_ADAPTER ex_body;
exception DATA_CONVERSION ex_body;
exception OBJECT_NOT_EXIST ex_body;
```

5. *Acronyms*

API	Application Program Interface
CIIF	Common Imagery Interoperability Facilities
CIIP	Common Imagery Interoperability Profile
CIIWG	Common Imagery Interoperability Working Group
CORBA	Common Object Request Broker Architecture
GIAS	Geospatial & Imagery Access Services
IASS	Image Access Services Specification
IDL	Interface Definition Language
ISO	International Standard Organization
NIMA	National Imagery and Mapping Agency
OGC	Open GIS Consortium
OMG	Object Management Group
TBD	To Be Determined
TBR	To Be Resolved
UCOS	USIGS Common Object Specification
UIP	USIGS Interoperability Profile
USIGS	United States Imagery and Geospatial System

6. Points of Contact

USIGS Architecture Integration Group
Ron Burns, National Imagery and Mapping Agency
Phone: (703) 808-0891
FAX: (703) 808-0531
Email: BurnsR@nima.mil

Joe Wesdock, National Imagery and Mapping Agency
Phone: (301) 227-3110 x428
Email: WesdockJ@nima.mil

Project Lead for MITRE Interface Definition Support
John Polger, National Imagery and Mapping Agency
Phone: (202) 863-3004
FAX: (202) 488-0271
Email: PolgerJ@nima.mil

NIMA Libraries Interface Definition
Charlie Green, Sierra Concepts, Inc.
Phone: (610) 347-0602
FAX: (610) 347-0602
Email: cpg.sci@mindspring.com

UCOS & GIAS Specifications & Support, and RFCs
Dave Lutz, The MITRE Corporation
Phone: (703) 883-7848
FAX: (703) 883-3315
Email: dlutz@mitre.org

USIGS Interoperability Profile (UIP)
Bill Nell, Lockheed Martin Management & Data Systems
Phone: (610) 531-6012
FAX: (703) 962-3698
Email: William.H.Nell@lmco.com